Procesamiento de Lenguaje Natural (NLP) – Introducción al Análisis de Texto

Contexto general

El **Procesamiento de Lenguaje Natural (NLP, por sus siglas en inglés)** es una rama de la inteligencia artificial que permite a las computadoras **entender, interpretar y generar lenguaje humano**.
A través de técnicas estadísticas, matemáticas y de aprendizaje automático, el NLP busca extraer significado y patrones de grandes volúmenes de texto.

En este notebook exploraremos cómo aplicar herramientas básicas de NLP utilizando Python y librerías populares como **NLTK**, **spaCy**, **Scikit-learn** y **pandas**.

Objetivos del proyecto

El propósito de este trabajo es comprender paso a paso cómo transformar texto sin procesar en representaciones numéricas que puedan ser utilizadas por algoritmos de Machine Learning.
A lo largo del notebook realizaremos las siguientes tareas:

1. **Carga y preprocesamiento de texto:** limpieza, normalización y tokenización.
2. **Análisis exploratorio del lenguaje:** conteo de palabras, frecuencia de términos y visualizaciones.
3. **Representación vectorial:** uso de técnicas como Bag of Words (BoW) o TF-IDF.
4. **Entrenamiento de un modelo de clasificación de texto:** por ejemplo, detección de sentimiento o categorización de documentos.
5. **Evaluación y métricas de desempeño** del modelo resultante.

Enfoque metodológico

1. **Preprocesamiento:**
    - Conversión del texto a minúsculas.
    - Eliminación de signos de puntuación, números y palabras vacías (*stopwords*).
    - Lematización o stemming según el caso.
2. **Vectorización del texto:**
    - Transformación de los documentos en vectores mediante **TF-IDF** o **CountVectorizer**.
3. **Modelado y evaluación:**

- Entrenamiento de modelos de clasificación (por ejemplo, Naive Bayes, Logistic Regression o SVM).
- Evaluación mediante métricas como *accuracy*, *precision*, *recall* y *F1-score*.

Resultado esperado

Al finalizar este notebook, se espera haber desarrollado un pipeline completo de análisis de texto, capaz de:

- Limpiar y procesar grandes volúmenes de texto.
- Representar el lenguaje en forma numérica.
- Entrenar un modelo predictivo sobre datos textuales.
- Visualizar e interpretar los resultados del análisis.

> **Nota:** El enfoque de este trabajo es educativo y demostrativo.
> Las técnicas aquí empleadas pueden adaptarse para tareas más complejas como análisis de sentimientos, clasificación de noticias, detección de spam o resumen automático de texto.

In [1]:
```
# 1️⃣ Librerías — IMDB (binario: positive/negative)

import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    ConfusionMatrixDisplay
)

import matplotlib.pyplot as plt

# Opciones/semilla
pd.set_option("display.max_colwidth", 120)
np.random.seed(42)

print("Librerías cargadas ✅")
```

```
Librerías cargadas ✅
```

2. Carga de datos desde CSV (usaremos solo los 80 primeros registros)

En esta sección se cargan los datos del archivo `IMDB Dataset.csv`, que contiene un conjunto de **50 000 reseñas de películas** junto con su **etiqueta de sentimiento** (*positive*

o *negative*).
Cada fila del dataset representa una opinión escrita por un usuario, la cual será utilizada para entrenar y evaluar un modelo de análisis de sentimiento.

Estructura del dataset original

El archivo incluye principalmente las siguientes columnas:

- `review` → texto completo de la reseña.
- `sentiment` → etiqueta de sentimiento (positivo o negativo).

Para simplificar y acelerar el desarrollo, en este ejemplo se trabajará con una **versión reducida de solo 80 registros**, suficientes para ilustrar el flujo completo del análisis NLP.

Pasos realizados en la celda

1. **Carga del CSV:**
   Se lee el archivo `IMDB Dataset.csv` usando `pandas` y se imprimen las columnas disponibles.

2. **Renombrado de columnas:**
   Las columnas se renombran a un formato más amigable:

   - `review` → `texto`
   - `sentiment` → `etiqueta`
3. **Traducción opcional de etiquetas:**
   Para mantener coherencia con el idioma del notebook, las etiquetas se traducen a español:

   - `positive` → `positivo`
   - `negative` → `negativo`
4. **Selección de columnas relevantes:**
   Se conservan únicamente las columnas `texto` (entrada) y `etiqueta` (salida o clase).

5. **Muestreo de registros:**
   Se toman los **primeros 80 registros** mediante `df.head(80)` para obtener una versión ligera y rápida de ejecutar.

6. **Verificación de distribución de clases:**
   Se imprime la cantidad de ejemplos positivos y negativos para asegurar que la muestra esté balanceada.

Resultado esperado

- El DataFrame final debe contener **80 filas** y **2 columnas** (`texto` , `etiqueta`).
- Las etiquetas traducidas deben mostrarse como `"positivo"` y `"negativo"` .

- Se despliega un vistazo de las primeras reseñas con `df.head()`.

> 🧠 **Nota:** Este subconjunto es solo con fines demostrativos; en un entorno real, el análisis se realizaría sobre la totalidad del dataset (50 000 reseñas) para mejorar la robustez del modelo.

In [2]:
```python
# 2️⃣ Carga de datos desde CSV (IMDB, 80 primeros registros)

# Ruta del CSV (ajusta si fuera necesario)
CSV_PATH = "IMDB Dataset.csv"

# Carga
df = pd.read_csv(CSV_PATH)

print("Columnas disponibles:", df.columns.tolist())

# Renombrar a nuestro esquema (texto/etiqueta)
# IMDB usa: review (texto), sentiment (positive/negative)
df = df.rename(columns={"review": "texto", "sentiment": "etiqueta"})

# Opcional: traducir etiquetas a español para mantener consistencia
df["etiqueta"] = df["etiqueta"].map({"positive": "positivo", "negative": "negati

# Selección y tipado
texto_cols = ["texto"]
label_cols = ["etiqueta"]
COL_TEXTO = texto_cols[0]
COL_LABEL = label_cols[0]

# Tomar solo 80 registros para versión corta
df = df[[COL_TEXTO, COL_LABEL]].dropna().head(80)
df[COL_TEXTO] = df[COL_TEXTO].astype(str)

print(f"Registros usados: {len(df)}")

# Distribución de clases
print(df[COL_LABEL].value_counts())

df.head(5)
```

```
Columnas disponibles: ['review', 'sentiment']
Registros usados: 80
etiqueta
negativo    44
positivo    36
Name: count, dtype: int64
```

Out[2]:

	texto	etiqueta
0	One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as ...	positivo
1	A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and g...	positivo
2	I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater...	positivo
3	Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fightin...	negativo
4	Petter Mattei's "Love in the Time of Money" is a visually stunning film to watch. Mr. Mattei offers us a vivid portr...	positivo

3. Preprocesamiento textual (simple y directo)

El preprocesamiento es una de las etapas más importantes en cualquier proyecto de **Procesamiento de Lenguaje Natural (NLP)**.
Antes de entrenar un modelo, el texto debe limpiarse para eliminar ruido y dejar solo la información relevante.

Objetivo de esta etapa

En esta versión simplificada:

- Utilizaremos la normalización interna del `TfidfVectorizer`, que:
  - Convierte automáticamente el texto a **minúsculas**.
  - Aplica una **tokenización básica**.
- Además, se realizará una **limpieza ligera previa** para eliminar etiquetas HTML y espacios innecesarios, ya que las reseñas del dataset IMDB pueden contener elementos como `<br />`, `<p>`, etc.

Pasos implementados

1. **Definición de la función `preprocesar_mínimo`:**
   - Elimina etiquetas HTML mediante expresiones regulares (`re.sub()`).
   - Reemplaza múltiples espacios consecutivos por un único espacio.
   - Aplica un recorte (`strip()`) para quitar espacios al inicio y final.
2. **Aplicación de la limpieza sobre el texto:**
   - Se crea una nueva columna `texto_proc` que contiene las reseñas ya preprocesadas.
   - Se muestra una vista previa de los primeros registros para confirmar el resultado.

Nota técnica

El preprocesamiento aquí es **intencionalmente liviano**, ya que el `TfidfVectorizer` manejará el resto de transformaciones internas como:

- Conversión a minúsculas.
- Tokenización y conteo de palabras.
- Eliminación de términos frecuentes y poco informativos.

En una versión más completa, podrían añadirse pasos adicionales como:

- Eliminación de *stopwords*.
- Lematización o stemming.
- Corrección ortográfica o normalización semántica.

Resultado esperado

- Los textos quedan listos para la etapa de **vectorización TF-IDF**.
- Se observa una reducción del ruido en el contenido (sin etiquetas HTML ni espacios excesivos).
- La estructura y significado de las reseñas se mantiene intacta.

In [3]:
```python
# 3️⃣ Preprocesamiento textual (simple y directo)

import re

def preprocesar_minimo(texto: str) -> str:
    """
    Limpieza ligera: elimina etiquetas HTML y espacios extras.
    (TF-IDF ya convierte a minúsculas por defecto)
    """
    texto = re.sub(r"<.*?>", " ", str(texto))  # quitar etiquetas HTML
    texto = re.sub(r"\s+", " ", texto)         # eliminar espacios múltiples
    return texto.strip()

# Aplicar limpieza mínima
df["texto_proc"] = df[COL_TEXTO].apply(preprocesar_minimo)
df.head(5)
```

Out[3]:

	texto	etiqueta	texto_proc
0	One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as ...	positivo	One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as ...
1	A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and g...	positivo	A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comfo...
2	I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater...	positivo	I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater...
3	Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fightin...	negativo	Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fightin...
4	Petter Mattei's "Love in the Time of Money" is a visually stunning film to watch. Mr. Mattei offers us a vivid portr...	positivo	Petter Mattei's "Love in the Time of Money" is a visually stunning film to watch. Mr. Mattei offers us a vivid portr...

4. Vectorización (TF-IDF) y 5. Modelo (Naive Bayes) — IMDb (positivo/negativo)

En esta sección combinamos dos pasos fundamentales del flujo de **Procesamiento de Lenguaje Natural (NLP)**: la **transformación del texto en vectores numéricos (TF-IDF)** y el **entrenamiento del modelo de clasificación** usando **Naive Bayes**.

---

1. Vectorización TF-IDF

El modelo de texto no puede trabajar directamente con palabras, por lo que primero se convierten en **representaciones numéricas** usando la técnica **TF-IDF (Term Frequency – Inverse Document Frequency)**.
Esta técnica asigna a cada palabra un peso que refleja su **importancia dentro del documento** en relación con el resto del corpus.

Características principales del proceso:

- Se utiliza la columna `texto_proc` (texto preprocesado).
- Se emplean **unigramas y bigramas** (`ngram_range=(1, 2)`) para capturar frases cortas como *"not good"* o *"very bad"*.
- Se limita el tamaño del vocabulario con `max_features=5000` para reducir ruido y complejidad.
- Se ignoran términos que aparecen en menos de 2 documentos (`min_df=2`).

---

2. Separación de datos

Antes del entrenamiento, los 80 registros se dividen en subconjuntos de:

- **70 % entrenamiento**
- **30 % prueba**

La partición es **estratificada (`stratify=y`)**, lo que asegura que la proporción de clases ("positivo" / "negativo") se mantenga igual en ambos subconjuntos.

---

3. Entrenamiento del modelo

Se implementa un pipeline con dos etapas consecutivas:

1. `TfidfVectorizer` → convierte el texto en vectores numéricos.
2. `MultinomialNB` → clasificador probabilístico Naive Bayes, ampliamente usado en tareas de NLP por su eficiencia y simplicidad.

El pipeline se entrena con `modelo.fit(X_train, y_train)` y queda listo para generar predicciones sobre nuevos textos.

Resultado esperado

- El modelo se entrena correctamente (✅ **"Modelo entrenado"**).
- A partir de ahora puede utilizarse para:
  - Predecir el sentimiento de nuevas reseñas.
  - Evaluar su precisión y métricas de desempeño (accuracy, F1-score, etc.).

> 💡 **En resumen:**
> Este paso transforma las reseñas textuales del dataset IMDb en una representación cuantitativa TF-IDF y entrena un modelo Naive Bayes que distingue entre opiniones **positivas** y **negativas**.

In [7]:
```
# 4️⃣ Vectorización (TF-IDF) y 5️⃣ Modelo (Naive Bayes) — IMDB

X = df["texto_proc"].values
y = df[COL_LABEL].values

# Separación en entrenamiento y prueba (70/30)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# TF-IDF con n-gramas (1,2) para capturar frases como "not good", "very bad"
modelo = Pipeline([
    ("tfidf", TfidfVectorizer(max_features=5000, ngram_range=(1,2), min_df=2)),
    ("clf", MultinomialNB())  # Clasificador probabilístico Naive Bayes
])

# Entrenamiento del modelo
modelo.fit(X_train, y_train)
print("Modelo entrenado ✅")
```

```
Modelo entrenado ✅
```

6. Evaluación y métricas — IMDb (positivo/negativo)

En esta etapa evaluamos el rendimiento del modelo usando los datos de prueba (`X_test` , `y_test`).

Métricas calculadas

- **Accuracy (Precisión global):** porcentaje total de aciertos del modelo.
- **Classification Report:** muestra métricas por clase, incluyendo:
  - *Precision:* proporción de predicciones positivas correctas.
  - *Recall:* proporción de verdaderos positivos detectados.
  - *F1-score:* promedio armónico entre precisión y recall.
- **Matriz de confusión:** visualiza qué cantidad de reseñas positivas/negativas fueron clasificadas correctamente o confundidas.

Interpretación

Estas métricas permiten analizar si el modelo:

- Generaliza correctamente a datos no vistos.
- Presenta sesgos hacia alguna clase (*positivo* o *negativo*).
- Está sobreajustado a los datos de entrenamiento.

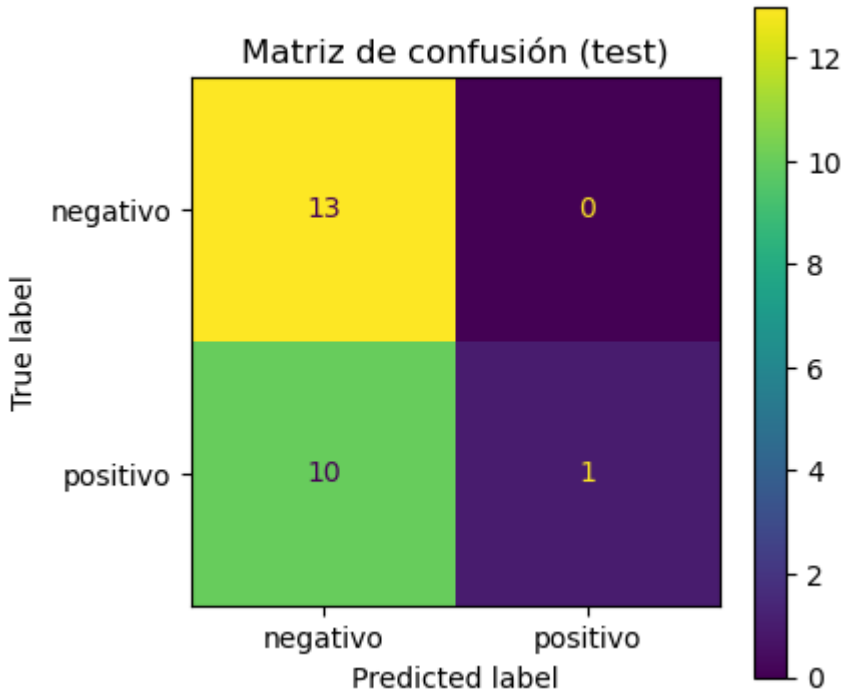
El gráfico de la matriz de confusión proporciona una vista rápida de los aciertos y errores, facilitando el análisis de los resultados del clasificador.

> **Consejo:** Si observas que las clases están desbalanceadas o el modelo tiende a predecir siempre la misma clase, considera ajustar los parámetros de TF-IDF o probar otro clasificador.

In [8]:
```python
# 6️⃣ Evaluación y métricas — IMDB

# Predicción en test
y_pred = modelo.predict(X_test)

# Métricas
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.3f}\n")

print("Classification report:")
print(classification_report(y_test, y_pred))

# Matriz de confusión (gráfico simple con matplotlib)
import numpy as np
cm = confusion_matrix(y_test, y_pred, labels=np.unique(y))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y))
```

```
fig, ax = plt.subplots(figsize=(4.5, 4))
disp.plot(ax=ax)
plt.title("Matriz de confusión (test)")
plt.tight_layout()
plt.show()
```

```
Accuracy: 0.583

Classification report:
              precision    recall  f1-score   support

    negativo       0.57      1.00      0.72        13
    positivo       1.00      0.09      0.17        11

    accuracy                           0.58        24
   macro avg       0.78      0.55      0.44        24
weighted avg       0.76      0.58      0.47        24
```



7. Visualización final atractiva

En esta sección se genera una **comparación visual** entre la distribución real y la distribución predicha de los sentimientos en el conjunto de prueba.

Objetivo

El propósito es visualizar de forma sencilla cómo el modelo se comporta en términos de cantidad de reseñas:

- **Reales:** etiquetas originales del dataset (`y_test`).
- **Predichas:** resultados estimados por el modelo (`y_pred`).

Descripción del gráfico

Se construye un **gráfico de barras** donde:

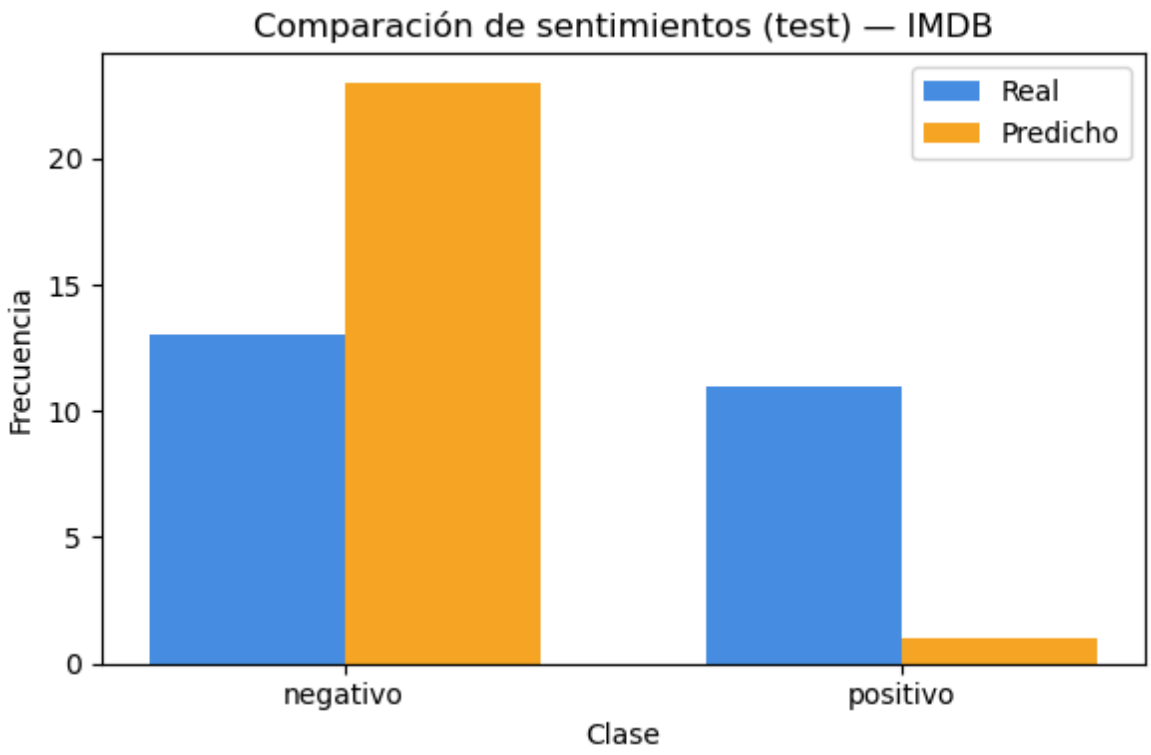
- Cada barra representa una clase de sentimiento (*positivo* o *negativo*).
- Las barras se agrupan para mostrar los valores **reales** y **predichos** lado a lado.
- Se utilizan colores diferenciados (`azul` para reales, `naranja` para predichos) para una comparación clara.

Interpretación

- Si las alturas de ambas barras (real vs. predicho) son similares, el modelo está clasificando de forma equilibrada.
- Diferencias notorias indican posibles errores sistemáticos (por ejemplo, tendencia a predecir demasiado una clase).

> 🧠 **Insight visual:** Este tipo de comparación es útil para detectar patrones de sesgo y validar el comportamiento global del clasificador antes de aplicarlo a un dataset más grande.

In [9]:
```python
# 7️⃣ Visualización final atractiva — Comparación Real vs Predicho (IMDB)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Distribución real y predicha
vals_true = pd.Series(y_test).value_counts().sort_index()
vals_pred = pd.Series(y_pred).value_counts().reindex(vals_true.index).fillna(0)

clases = vals_true.index.astype(str)
x = np.arange(len(clases))
width = 0.35  # separación entre barras

# Gráfico de barras
fig, ax = plt.subplots(figsize=(6, 4))
b1 = ax.bar(x - width/2, vals_true.values, width, label="Real", color="#4a90e2")
b2 = ax.bar(x + width/2, vals_pred.values, width, label="Predicho", color="#f5a6

ax.set_title("Comparación de sentimientos (test) — IMDB")
ax.set_xlabel("Clase")
ax.set_ylabel("Frecuencia")
ax.set_xticks(x)
ax.set_xticklabels(clases)
ax.legend()
plt.tight_layout()
plt.show()
```



Predicción rápida (demo)

En esta última parte realizamos una **prueba interactiva** del modelo, ingresando frases nuevas para observar la predicción de sentimiento en tiempo real.

Objetivo

Probar si el modelo entrenado logra identificar correctamente el tono emocional de nuevos comentarios, incluso aquellos no presentes en el dataset original.

Funcionamiento

1. Se define una función `predecir(texto)` que:
   - Aplica la misma **limpieza mínima** usada en el entrenamiento (`preprocesar_minimo`).
   - Transforma el texto y ejecuta `modelo.predict()` para obtener la clase.
   - Muestra el texto original y su predicción (POSITIVO o NEGATIVO).
2. Se prueban ejemplos de reseñas típicas en inglés (como las del dataset IMDb).
3. Se puede modificar la variable `nuevo` para probar comentarios personalizados.

Interpretación de resultados

- El modelo devuelve una predicción textual, por ejemplo: `Predicción: POSITIVO` o `Predicción: NEGATIVO` .

- Si todos los ejemplos tienden a la misma etiqueta, podría ser necesario:
  - Aumentar la cantidad de datos de entrenamiento.
  - Revisar el equilibrio de clases o ajustar los parámetros de TF-IDF.

> 💬 **Conclusión:** Este bloque demuestra de manera práctica cómo utilizar el modelo entrenado para clasificar nuevos textos y validar su comportamiento en escenarios reales.

In [10]:
```python
# 💉 Predicción rápida (demo) — IMDB (positivo/negativo)

def predecir(texto: str):
    # usar la misma limpieza mínima que en entrenamiento
    t_proc = preprocesar_minimo(texto)
    pred = modelo.predict([t_proc])[0]
    print("💬 Comentario:", texto)
    print("🧠 Predicción:", pred.upper(), "\n")

# Cambia/añade ejemplos (IMDB está en inglés)
ejemplos = [
    "The movie was amazing, I loved the performances and the story.",
    "Terrible film. Boring, predictable, and way too long.",
    "It was okay, some parts were good but overall disappointing.",
]

for e in ejemplos:
    predecir(e)

# Ejemplo único
nuevo = "The plot was weak and the acting was bad."
predecir(nuevo)
```

```
💬 Comentario: The movie was amazing, I loved the performances and the story.
🧠 Predicción: NEGATIVO

💬 Comentario: Terrible film. Boring, predictable, and way too long.
🧠 Predicción: NEGATIVO

💬 Comentario: It was okay, some parts were good but overall disappointing.
🧠 Predicción: NEGATIVO

💬 Comentario: The plot was weak and the acting was bad.
🧠 Predicción: NEGATIVO
```